Programming for Games: Design of an Input Algorithm

There are two methods of input that I considered when designing my algorithm, firstly the more modern method that is used in *Emily Is Away* (2015), *Avast, Ye Mateys!* (2015) and *Coming Out Simulator 2014* (2014). These games all present the player with a list of available options specific to the current room or time. This method however does not pose as much of an interesting or challenging task to undertake when handling the user’s input and therefore I decided against this method.

The second and more traditional method is providing the player with a set of commands they can use and letting them construct their own instructions, the classic game *Zork* (1979) executes this fantastically allowing the player to use full sentences as if they were speaking to another person yet still allowing more simple commands like the two-word command system in the original text adventure *Colossal Cave Adventure (1977)*. *Deeper* (2016) is similar to Zork because it allows the player to use natural English sentences, however due to it being made for web browsers it allows the player to also use a GUI to navigate the game alongside the text prompt. I was thoroughly impressed by the simplicity and ease this presented the player so I decided that I would attempt to re-create this form of input handling in my algorithm.

Firstly, I split the player’s input into a list of separate words and removed all of the conjunctions, prepositions and other unnecessary words from that list to leave me with only the verbs and nouns (such as item names and exits). A *for loop* running through a list of “*removable*” words was used to remove all of the conjunctions and prepositions that I had defined in the class. Later in the development of the algorithm, I wanted the items in the game to be able to have names containing spaces like in the games I had researched. This added another step to the string formatting method which involved stitching individual strings together between certain “*splitter*” words such as “with” and “and”. Once the strings are all stitched back correctly the *splitter* words are all removed the same way as the *removable* words were.

After formatting, I am left with a list of strings that can contain verbs, item names and exits, I can now start to figure out what the player wants to do and from this, attempt to call the right method for them.

The first check I make is to see whether there are more than one words in the command list. If there is only one word, I firstly check to see if it’s an exit in the current location, if so then the “moveToLocation” method is called and I return out of the method. However, if the command wasn’t a direction then it is put into a switch statement checking if it’s a look, show inventory or a help command, calling their own methods respectively.

Commands containing more than one word go through the *Complex command* part of the algorithm. Here I check to find a verb and set the correct command type for that verb, a dictionary is used here to pair string representations of the verbs to their own enumerated versions, this allow me to easily allow multiple similar verbs to call the same method such as “move” and “walk” both setting the *Move* command type. Once the command type has been found, it is used by another switch statement to pass the remaining strings in the *command list* as arguments in the matching methods.

The idea of searching for a verb in the input comes directly from Zork which, if it cannot find one will tell you directly it doesn’t understand the verb you entered.

The move method is split up into two methods to allow an input of a string as well as the exit’s direction, the method taking in the string eventually calls the other one I’ll explain that one first. Firstly, I am looking for an exit in the current location matching the given string, this is done by using the List<>.Find method with a lambda expression which compares all of the *ToString* values of the elements to the given string. Before learning about lambda expressions I had been using *foreach* loops to go through the list and individually check the names against the string which became rather messy, the way I search for the exit now saves the method from getting too complicated and messy. Finally, once the exit is found we pass it into the other move method; we check that the exit is not locked, by seeing if the exit’s Key property is null. Now all I do is set the *current location* to the exit’s location and return out of the method; the game then updates and displays the new location.

The use command allows the player to unlock an exit with a Key item they have picked up, to allow the use of keys in the game I added a key variable to the Exit class where if an item is present, the exit is locked and alternatively, a null item means that the exit is unlocked.

Two things are required when using items, the item you wish to use and the exit you want to use it on. To find the exit I use the same method as in the move method however the item is found by using a similar predicate in the find method however to ensure that the item is a key, I use the *as* keyword so that the algorithm will only be able to progress if both the exit and the key are not null; a regular item will return null when casting it to a key. Now all that is left is checking that the found key is equal to the exit’s key and set it to null if it is, unlocking the exit.

All of the other commands such as taking and inspecting items use the same principals as the move and use commands, regarding finding the items in inventory lists.

# Bibliography

The Pixie. (2016). *Deeper*. [Video Game]. Available at: <http://textadventures.co.uk/games/view/em15b32xd0o-y-ysvgrtcg/deeper> [Accessed 24 Feb 2016]

Anderson, T., Blank, M., Leibling, D. and Daniels, B. (1979). *Zork*. [Video Game] Infocom. Available at: <http://textadventures.co.uk/games/view/5zyoqrsugeopel3ffhz_vq/zork> [Accessed 24 Feb 2016]

Crowther, W., Woods, D. (1977) *Colossal Cave Adventure*. [Video Game] CRL. Available at: <http://www.amc.com/shows/halt-and-catch-fire/exclusives/colossal-cave-adventure> [Accessed 25 Feb 2016]

Textadventures.co.uk, (2016). *textadventures.co.uk - Create and play text adventure games*. [online] Available at: http://textadventures.co.uk/ [Accessed 25 Feb. 2016].

Case, N (2014). *Coming Out Simulator 2014.* [Video Game]. Available at: <http://textadventures.co.uk/games/view/fj9sjxcci02r6gioefxd7a/coming-out-simulator-2014> [Accessed 25 Feb 2016]

Seeley, K. (2015). *Emily Is Away* [Video Game] Steam. Kyle Seely.

Peter123. (2015) *Avast, Ye Mateys!* [Video Game]. Available at: <http://textadventures.co.uk/games/view/gb5rrcs-8u_5fxbvwbtf8a/avast-ye-mateys> [Accessed 25 Feb 2016]